Satisficing Q-Learning: Efficient Learning in Problems with Dichotomous Attributes

Michael A. Goodrich and Morgan Quigley
Computer Science Department, Brigham Young University

Abstract

In some environments, a learning agent must learn to balance competing objectives. For example, a Q-learner agent may need to learn which choices expose the agent to risk and which choices lead to a goal. In this paper, we present a variant of Q-learning that learns a pair of utilities for worlds with dichotomous attributes and show that this algorithm properly balances the competing objectives and, as a result, efficiently identifies satisficing solutions. This occurs because exploration of the environment is restricted to those options which, according to current knowledge, are likely to avoid unjustifiable exposure to risk. We empirically validate the algorithm by (a) showing that the algorithm quickly converges to good policies in several simulated worlds of various complexities and (b) applying the algorithm to learning a force feedback profile for a gas pedal that helps drivers avoid risky situations.

1 Introduction

Q-learning is a very successful algorithm, partly because it assumes very little about the world but still produces effective solutions. Ideally, the Q-learning algorithm produces a set of Q-values that converge to the expected discounted utility for selecting an action in a given state of the world. When convergence occurs, an optimal solution from a given state can be chosen by selecting the action that maximizes the Q-value. However, to guarantee that the estimated Q-values truly converge, it is necessary to explore all possible state-action combinations an infinite number of times. Since this is clearly impossible, using Q-learning in practice is restricted to converging with high probability to Q-values that correspond to acceptable policies.

In addition to the difficulty imposed by the convergence requirements, conventional Q-learning also requires that a reinforcement structure be created that balances rewards and penalties. To use conventional Q-learning, the numerical values of rewards and penalties must be selected such that penalties are not so high that they mask rewards, and vice versa. This often requires a tuning process wherein reward values, penalty values, and discount values are iteratively adjusted by a designer until a functional balance is achieved.

We present an extension to Q-learning that quickly converges to good enough policies that balance competing objectives in worlds with dichotomous decision attributes. For many such worlds, the reinforcement structure is such that there are many more choice sequences that lead to a penalty than that lead to a reward. Consequently, penalties often propagate through the world more rapidly than rewards, which means that it is often easier to learn what is risky than what is right. If proper balance between rewards and penalties can be achieved, the rapid propagation of penalties can be used to make the algorithm converge quickly. This line of reasoning can be reversed to help an agent efficiently learn penalties in a reward-rich environment.

The algorithm essentially keeps two Q-tables: one that encodes information about rewards and one that encodes information about penalties. This algorithm then combines these dichotomous attributes using two decision principles: satisficing and non-domination. The resulting algorithm generates an exploration policy that efficiently avoids costly actions while trying to find a reward. We empirically validate the algorithm by (a) showing that the algorithm quickly converges to good policies in a several simulated worlds of various complexities and (b) applying the algorithm to learning a force feedback profile for a gas pedal that helps drivers avoid risky situations.

2 Relevant Literature

The efficiency claims of this paper are similar in spirit to fast and online reinforcement learning techniques like DYNA-Q [13, 9], fast value iteration [14], and Sarsa Q-learning [13]. The primary differences are that no explicit information of transition probabilities is used, and no balancing of rewards is required by the designer. Of the work in online learning, the most directly similar to our approach is [7] which also uses the satisficing principle, albeit an aspiration-based rather than a dichotomy-based definition of satisficing. The notion of satisficing has been used in making decisions in other domains as well. These include robust

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Satisficing Q-Learning: Efficient Learning in Problems with Dichotomous Attributes	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Brigham Young University,Computer Science Department,33361 Talmage Building,Provo,UT,84602**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
see report

15. SUBJECT TERMS

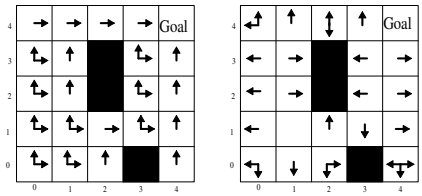
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**8**	

decision-making [8], search [11], and robust control [1]. Satisficing has proven useful in each of these contexts because of the ability to suspend judgement between alternatives, and the satisficing Q-learning algorithm exploits this ability to guide efficient exploration.

3 Satisficing Q-Learning

As mentioned in the introduction, we have created a variation of Q-learning that seeks to find a useful balance between rewards and penalties. We adopt the standard Q-learning formalization, which assumes that the world consists of a state, θ, selected from a set of possible states; the learning agent must choose an action, a, from a set of possible actions, in some intelligent way. Furthermore, the world is assumed to have a first-order Markov transition probability. We use the term *reinforcer* or *reinforcement structure* to denote the pattern of numerical feedback created by the designer to reward or penalize the agent, and the terms *reward* and *penalty* to denote positive and negative reinforcers, respectively.

In penalty-rich or reward-rich worlds, this algorithm is designed to find this balance efficiently. To motivate how this algorithm accomplishes this, consider the world depicted in Figure 1. (Although we motivate the algorithm for penalty-rich worlds, note that a similar argument holds in reward-rich worlds.) In this world, the objective is to safely navigate



(a) (b)

Figure 1: A simple world with the (a) best goal-achieving actions shown, and (b) worst risk-exposing actions shown.

through the world and reach the goal. For each cell, the best actions (those that produce minimal length paths) and worst actions (those that cause collisions) can easily be identified. These actions are shown in Figure 1 (a) and (b), respectively.

Consider the cell located three from the right and two from the bottom. Clearly, the agent should not move up into the wall but instead move to the right toward the goal. However, if there is randomness in the world, an attempt to move to the right may instead result in an upward movement into the wall. If collisions were unimportant, then the agent would have no problem moving to the right, but if collisions are costly than the agent may think twice before attempting to squeeze past the wall. Thus, there is a tension between seeking the goal reward and avoiding the collision penalty. Our algorithm proposes a technique for finding a balance between such competing objectives.

For the world in Figure 1, it is fairly easy to determine which actions are costly because such actions immediately lead to collisions. In fact, in all but one cell, there is at least one action in each cell which is obviously bad. Exploration effort should not be spent repeatedly trying these actions because only one or two experiments will confirm that they are indeed costly. This means that the number of actions that must be explored in each state can be reduced, especially in the lower right corner where only one action deserves serious consideration.

3.1 Satisficing

The term satisficing was used by Simon in decision-making contexts to mean any decision that is "good enough" [12]. The notion of satisficing has also been used to identify "good enough" actions in worlds with competing objectives [5, 4]. Under this characterization, any action is justified provided that the expected reward of that action exceeds its expected cost. This assumes, of course, that the units of expected reward and expected cost can be expressed in commensurate units. Using this notion of satisficing, we can completely characterize satisficing-based choice in worlds with dichotomous goals as the set of actions whose expected reward exceeds the expected penalty. For a given state θ, this is given by $S(\theta) = \{a : \mu_A(a, \theta) \geq \mu_R(a, \theta)\}$ where the subscript A indicates what is acceptable, meaning the degree to which an action tends to achieve rewards; and the subscript R indicates what is rejectable, meaning the degree to which an action tends to lead to penalties. This approach requires us to learn both the *acceptability*, μ_A, of a state-action pair as well as the *rejectability*, μ_R, of the same pair.

Suppose that it is possible to (a) learn μ_A and μ_R independently and (b) guarantee that μ_A and μ_R are comparable. (Since utilities are unique only up to a positive affine transformation [10], making utilities comparable requires the selection of a common scale and zero-point.) Under these conditions, it is possible to identify $S(\theta)$ and use this set as the basis for exploring the world. To explore, we can justifiably choose any $a \in S(\theta)$ and then use the resulting experience to update μ_A and μ_R. If, for example, the environment is penalty-rich, then μ_R will quickly take shape which means that exploration will quickly learn to avoid obviously bad choices.

3.2 Optimization and Non-Domination

It is reasonable to consider what would happen if we selected actions that maximized a weighted sum of μ_A and

μ_R. Multi-attribute utility theory addresses the problem of maximizing the weighted sum of utilities when the weights are unknown by using the principle of non-domination. As it applies to μ_A and μ_R, an action is dominated if another action exists that has higher acceptability and lower rejectability. If we denote the set of actions that dominate a particular action a for a given state θ by $\mathcal{N}(a, \theta)$ (see [5] for a formal definition), then we can combine the satisficing set with the set of non-dominated actions to refine the set of justifiable actions. Formally, the *strongly satisficing set* is defined as the set of non-dominated satisficing options [5]

$$\mathcal{S}(\theta) = \{a : a \in S(\theta) \textbf{ and } \mathcal{N}(a, \theta) = \emptyset\}. \quad (1)$$

Elements in this set are satisficing (the expected reward outweighs the expected penalty) and non-dominated (no other action has both higher expected reward and lower expected penalty).

3.3 Reinforcer Structure

We now turn attention to learning μ_A and μ_R using a variant of Q-learning, and guaranteeing that they are comparable. The Q-learning algorithm updates estimates of the quality of a state-action pair as follows:

$$Q(a, \theta) \leftarrow (1 - \alpha)Q(a, \theta) + \alpha(r(a, \theta) + \gamma \max_{v'} Q(v', \theta'))$$

where α is the learning rate, γ is the temporal discount factor, θ' is the state that results from choosing action a in state θ, and $r(a, \theta)$ is the (possibly stochastic) reinforcer function.

We impose a reinforcer structure where $r(a, \theta) > 0$ means that an action has reached a goal and where $r(a, \theta) < 0$ means that an action caused the agent to do something undesirable. We set $r(a, \theta) = 0$ as the point that represents the status quo; as such, rewards are greater than zero and penalties are less than zero. Note that restricting the reinforcer in this way is one step toward making μ_A and μ_R comparable since we have identified a common zero-point. (Choosing a common scale will be discussed in a subsequent section.) Rather than relying on designer judgment to select balanced numerical values for rewards and penalties, we instead use a variant of the conventional Q-learning algorithm.

3.4 Rewards and Penalties

The Q-learning algorithm can be altered to encode only the (best estimate of the) expected reward for taking an action in a particular state of nature. We use the standard Q-learning update but only allow rewards (not penalties) to influence the estimate. We call this new function the G-function, as in Goal-function. The learning rule is

$$G(a, \theta) \leftarrow (1-\alpha)G(a, \theta)+\alpha(\max(0, r(a, \theta))+\gamma G(v', \theta')). \quad (2)$$

In addition to rewards, there are also penalties for making bad decisions. If we let $L(a, \theta)$ (L for *Loss*) encode only penalty-based information, then a high $L(a, \theta)$ indicates a high expected penalty whence accepting a gives a cost of $L(a, \theta)$. L can be updated by considering only penalties (negative reinforcers) as follows:

$$L(a, \theta) \leftarrow (1-\alpha)L(a, \theta)+\alpha(\max(0, -r(a, \theta))+\gamma L(v', \theta')). \quad (3)$$

Since $r(a, \theta)$ returns a positive (reward) value to reinforce goal-achieving actions and a negative (penalty) value to punish fault-achieving actions, we use the *max* operator so that only positive values of $r(a, \theta)$ influence the G-function and so that only negative values of $r(a, \theta)$ affect the L-function. In both equations, v' is an action selected by a technique that we will specify later (Equation (6)).

3.5 Acceptability and Rejectability

Recall that $S(\theta)$ requires that μ_A and μ_R be comparable; that is, they must have the same scale and the same zero point. We have constructed G and L such that they have the same zero point, and in this section we use them to give μ_A and μ_R the same scale. Satisficing Q-learning requires the agent to divide one unit of value across all possible actions. Then μ_A and μ_R are defined as

$$\mu_A(a, \theta) = \frac{G(a, \theta)}{\sum_a G(a, \theta)} \quad (4)$$

$$\mu_R(a, \theta) = \frac{L(a, \theta)}{\sum_a L(a, \theta)} \quad (5)$$

where the sum is taken over possible actions. Consider how this normalization would work in the world in Figure 1. For the cell in the lower right corner, dividing one unit of value across μ_A and and one unit across μ_R allows only one action to satisfice: $a = Up$. Thus, normalizing in this way allows us to identify actions whose reward-seeking ability, relative to other actions, outweighs exposure to penalties, relative to other actions. At every state, choices are made relative to the set of rewards and penalties that can be expected from this state.

3.6 Trembling L

While learning, the G and L estimates will experience transients that prohibit accurate identification of the "true" satisficing set. It is therefore possible for the agent to prematurely eliminate an action from exploration if the outcome of a particular experiment produces an initially undesirable outcome. A solution to this problem is to introduce "tremble" thereby causing a sufficient amount of evidence to mount before an action is pruned [6]. Although trembling does not completely prevent sub-standard outcomes, it does

make them unlikely. Thus, we tremble L for all actions as $L(a,\theta) \leftarrow L(a,\theta) + \eta$ where η is uniformly distributed on the interval $[0, H]$ and where H is set empirically. Note that this tremble is on the "upside" meaning that it can only inflate the value of L.

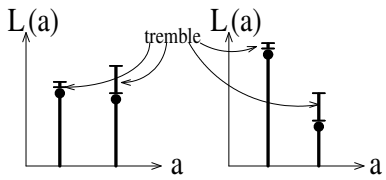


Figure 2: Two possible L function for two actions with tremble. Trembles are exaggerated for purposes of explanation.

The effects of tremble are illustrated in Figure 2. On the left, an L-function is trembled when the L-values are very close together. This causes a switch in which action is most rejectable. On the right, a different L-function is altered with the same amount of tremble. Because the L-value of the first action is so much higher, the tremble does not change which action is most rejectable. Thus, trembles serve to drive the estimated values of L higher than that obtained by experience and helps prevent premature pruning of potentially desirable actions.

If tremble persists throughout learning, it will eventually overcome the learned values of L causing, in effect, the agent to forget what it had learned about negative consequences. This is especially problematic since the decay on the learning rate, α, prevents new observations from altering L which means that the tremble soon comes to dominate μ_R. To counter this, the tremble magnitude is decayed exponentially over time, $H_t = H \times 0.999^t$, where t is the trial number. A value of $H = 0.001$ and a discount rate of 0.999 was chosen empirically for our experiments.

3.7 The Algorithm

The complete algorithm is given in Figure 3. To complete

1. Initialize $G = L = 1$ and set $H = 0.001$.
2. Repeat steps 3-5 until goal is reached.
3. Calculate μ_A and μ_R from G and L.
4. Apply a from $\mathcal{S}(\theta)$ with a uniform probability.
5. Update G and L, and tremble L.
6. Decay tremble and learning rate.
7. Repeat steps 2-6 a fixed number of trials.

Figure 3: The satisficing Q-learning algorithm.

step 5, we must decide what future values of G and L will be used to update $G(a,\theta)$ and $L(a,\theta)$ in (2)-(3). Future work should explore how best to choose this action. In this paper, we use the observation that the action

$$v' = \arg \max_{v \in S(\theta')} \Big[\mu_A(v,\theta') - \mu_R(v,\theta')\Big] \qquad (6)$$

is guaranteed to be strongly satisficing [5] and is therefore a reasonable heuristic estimate of the choice likely to be made in the next state.

4 Simulation Results

To quantify how Satisficing Q-learning balances competing objectives and facilitates efficient exploration, we compared the algorithm against other Q-learning algorithms in a series of simulation experiments on worlds with a varying number of obstacles. A reward of $r(a,\theta) = 1.0$ was awarded each time the agent reaches the goal, and a penalty of $r(a,\theta) = -1.0$ was assessed each time the agent hits a wall. Each world had $16 \times 16 = 256$ possible states with obstacles placed randomly throughout the world. Several obstacle densities were chosen, with the number of obstacles rangeing from 5 to 65. Ten different worlds were created at each obstacle density.

Obstacle placement was restricted to ensure that no two obstacles occupy the same cell. Furthermore, no obstacles were located in the goal or start cells, located in cell (16,16) and (1,1), respectively. Finally, the obstacles were not be placed in a way that prevented the agent from reaching any open cell from any other open cell; this prevented an agent from starting inside an area surrounded by obstacles and never reaching the goal.

Because we wish to evaluate how the algorithm balances competing objectives and how efficiently it learns, two measures of learning efficiency are presented: average path length per epoch and average number of collisions per epoch. On the test trials, the satisficing Q-learning algorithm chose an action randomly from $\mathcal{S}$, and the Soft-Max Q-learning algorithm chose the action with the maximal Q-value (i.e., Q-learning explored on learning trials, but exploited on test trials).

4.1 Selecting Parameters

Actions. The agent must choose from the set of compass directions, {N,S,E,W}. The agent usually moves in the direction chosen, but with some probability, denoted *rc*, the agent moves in a direction uniformly distributed among the other three directions. For example, when *rc*$= 0.6$, the agent moves in the chosen direction 60% of the time and in one of the other three directions 13.3% of the time.

Epochs. Learning epochs began at a random starting points and proceed until the goal was reached. Every 100 trials, the agent was placed at the starting location and chose according to its policy until it reached the goal. On these test trials, the number of collisions and path length were measured. Data were then averaged over 1000 Q-learners trained from uniform initial Q-values.

Discount Rate. To help determine the discount rate, we ran a series of experiments in a carefully designed 16×16 world with an irregularly placed wall in the center of the world. This wall forces the agent to travel through one of two narrow corridors to reach the goal. When $\gamma \leq 0.9$ and *rc*< 0.70, conventional Q-learning learns that it is best to stay in the southwest corner rather than risk passageway to the goal. The probability of hitting a wall is high enough that the agent learns to stay in a "safe zone" where collisions are avoided but where the goal is never reached. To avoid this effect, experiments are reported for $\gamma \geq 0.95$.

Learning Rate. For each algorithm, the learning parameter α is initialized to 0.5 and decayed as

$$\alpha_i = \max\left\{\frac{1.0}{i+2.0}, \frac{0.25}{1.0+e^{(i-10000)/2000}}\right\}$$

where i is the epoch number. This decay function begins with a sigmoidal decay which aggressively learns during the first epochs, and when the sigmoid function gets small learning decays linearly. The magic numbers 10,000 and 2,000 are set so that the sigmoid is centered at 5,000 epochs and decays suitably fast. These tuning values were hand selected because experiments demonstrated that they gave conventional Q-learning a good chance of learning an efficient path.

4.2 Single World Results

In this section, we use a handcrafted world to compare the Satisficing algorithm to the standard Q-learning with two exploration strategies: pure exploitation and Softmax with a probability of exploiting what is known set to 0.925. Data was gathered for *rc* $\in \{0.6, 0.9, 0.99\}$ using $\gamma = 0.95$.

Figure 4 shows the average number of collisions as a function of epoch for *rc*=0.99 and illustrates the main ways that satisficing Q-learning differs from conventional Q-learning. In the figure, the satisficing algorithm reaches a point of minimal collisions faster than the other two exploration algorithms and reaches a smaller number of collisions too.

Average steady state values for each experiment, shown below, support this conclusion.

Condition		*rc*		
		0.6	0.9	0.99
Path Length	Sat	100	50	35
	Soft	2100	40	35
	Exp	1000	40	35
# of Collisions	Sat	20	0.3	0.03
	Soft	100	0.4	0.1
	Exp	45	0.2	0.03

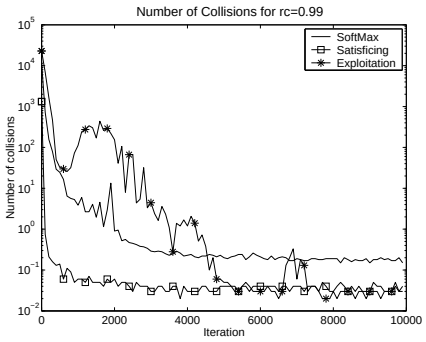


Figure 4: Average number of collisions for *rc*=0.99 for pure exploitation, softmax exploration, and satisficing exploration. In this world, exploitation eventually matched the number of collisions produced by satisficing, but this is usually not true for other worlds or values of rc.

The satisficing algorithm tends to find short paths that minimize collisions as well as or better than the other approaches. Moreover, the trend in Figure 4 holds where the satisficing algorithm reaches the steady state more rapidly than the other approaches in all but one experimental condition. This data suggests that the satisficing Q-learning algorithm efficiently learns to balance goal-seeking and avoiding collisions. We present more data to support this argument in the next section.

4.3 Quantitative Results

Data was gathered for *rc*$\in \{0.6, 0.9, 0.99\}$, but in the interest of space only representative results are shown in Figures 5-7. Note that these data are given for $\gamma = 0.995$ because lower values of γ tended to cause the SoftMax approach to enter into safe zones too frequently. Error

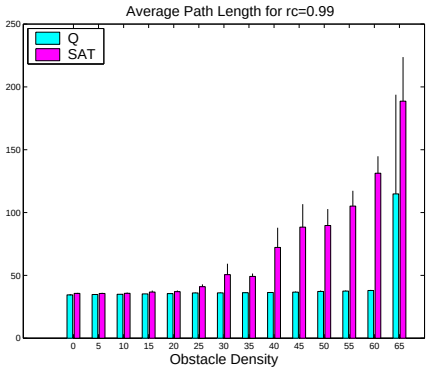


Figure 5: Average path lengths for *rc*=0.99 as a function of obstacle density.

bars represent one standard deviation computed over the ten

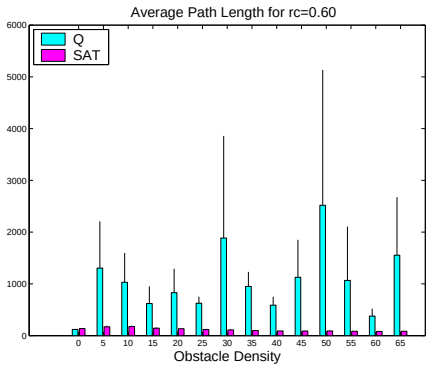


Figure 6: Average path lengths for *rc*=0.60 as a function of obstacle density.

worlds for each obstacle density. For *rc*=0.60 with 30 obstacles, results for Soft-Max are reported for only nine of the ten worlds — on one of the worlds, the algorithm failed to find a solution to the goal; the satisficing Q-learner results are shown for all ten worlds. The results for *rc*=0.90 are omitted because they do not reveal anything not shown in the other two plots.

Two observations are important. First, the satisficing algorithm and the Q-learning algorithm balance goal-seeking and obstacle-avoiding differently. This is easiest to see by considering *rc*=0.99. Under these conditions, the Q-learning algorithm finds a shorter average path than satisficing Q-learning, but collides nearly twice as often for each obstacle density. This indicates that the conventional Q-learner places a higher relative importance on goal seeking than obstacle avoiding under conditions of near certainty when $\gamma = 0.995$.

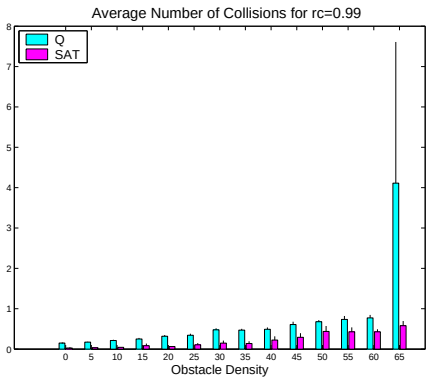


Figure 7: Average number of collisions for *rc*=0.99 as a function of obstacle density.

This becomes problematic as randomness increases. Consider, the extremely long paths, high number of collisions, and variability between experiments of the Soft-Max Q-learner for *rc*=0.60. These results indicate that this agent does not consistently find a proper balance between goal-seeking and obstacle-avoiding, at least for this value of γ. Instead, there is a tendency of the algorithm to find a "safe zone" and sit there until randomness pushes the agent through risky regions. In contrast, Satisficing Q-learning appears to more consistently balance goal-seeking and obstacle-avoiding across a wide variety of world conditions for a fixed set of parameter values.

The second observation is that the Satisficing Q-learning agent converges to a solution more quickly than the Soft-Max agent. To quantify this, we estimated convergence time by identifying the *convergence point* for each world. To understand how we did this, consider the plots of the number of collisions shown in Figure 4. These plots indicate that the agent has learned a consistent solution by the last iteration of the learning algorithm (partly because of α-decay). We use this observation to compute the convergence point by taking the average path lengths and number of collisions over the last 10 test trials for each value of *rc* and for each obstacle density. We say that the algorithm has *approached the convergence point* if a running average of five samples is within some percentage of the convergence point for both number of collisions and path length. Because larger values of *rc* introduce more randomness into the data, we use different percentages of the convergence point to estimate algorithm convergence. For *rc*=0.99, the percentage was 20%, for *rc*=0.9, the percentage was 30%, and for *rc*=0.6 the percentage was 40%. The average convergence times, shown

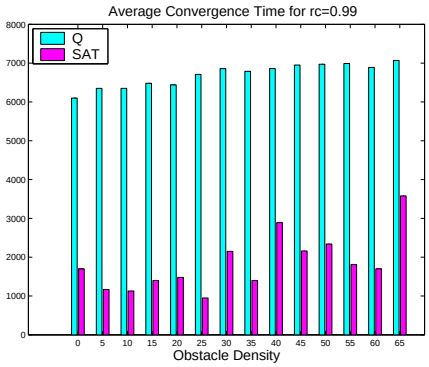


Figure 8: Average convergence times for random coefficient of 0.99 for softmax exploration and satisficing exploration as a function of obstacle density.

in Figure 8, show that even though both Soft-Max and Satisficing use the same learning decay rate, the Satisficing algorithm consistently reaches convergence more than twice as fast for all values of *rc*. These results indicate a greater degree of efficiency for the Satisficing algorithm, and support the hypothesis that balancing the competing objectives can make learning more efficient in penalty rich worlds.

4.4 Is it Satisficing or Normalizing?

It is interesting to ask the question whether it is the satisficing exploration or the normalization that makes convergence happen more quickly. To test this, we conducted an experiment where L and G were learned separately and then normalized to produce μ_R and μ_A. We then did soft-max exploration on $\mu_A - \mu_R$ rather than strongly satisficing exploration. For each value of *rc*, the satisficing algorithm produces fewer collisions more quickly than the normalized algorithm. This suggests that it is not just normalization of payoffs and costs that produces the benefit, but also the use of satisficing and non-domination to guide and regulate exploration.

5 Gas Pedal Example

The goal of this section is to describe parameter settings and results from applying the algorithm to the design of a force feedback gas pedal that will reduce driver risk without increasing driver workload. This problem is a practical example of how learning can efficiently occur in a reward-rich environment when penalties are rare.

5.1 Selecting Parameters

In this section, we discuss the types of actions, rewards, penalties, and states that were used to learn force feedback to support longitudinal control.

States. The state representation requires dimensions that could indicate deviation from the ideal driving behavior. This was implemented by noting that ideal driver following has infinite time to collision and a time headway on the order of 1.5 to 2 seconds [2]. This state space was discretized into nine headway values and nine inverse time to contact values.

Actions. Three forces were considered, corresponding to a large extra force toward the neutral position, a small extra force toward the neutral position, and no extra force above that provided by a passive return spring. When an action was selected, it was applied for at least 0.66 seconds during training or 0.25 seconds during testing and lasted until the state changed. (The difference in the times serves to emphasize the consequences during training, and helps speed information propagation.)

Rewards. Intuitively, an action is good if it enhances comfort and bad if it allows the driver to be exposed to risk. We selected two first order estimates of comfort, workload and impedance, which were estimated using techniques described in [3]. Subjective thresholds were then set so that actions were rewarded if both workload and impedance were low.

Penalties. An action is bad if it exposes a driver to risk. We penalize an action if it leads to a collision.

Learning Rate. The learning rate α was not decayed, but was rather set to a fixed value of $\alpha = 0.1$ For environments where it is difficult to estimate the speed with which information propagates through the state space, selecting a constant but small learning rate allows information to be propagated between states and prevents premature convergence, but at the cost of generating noisy Q-values.

Training. Training the algorithm consisted of starting with a uniform Q-table. The training driver passively drove the vehicle for fifteen minutes will the algorithm quickly learned identify the consequences of actions. The agent quickly learned to avoid impeding the trainer's pedal actions by collecting high rewards when it did not increase the pedal resistance. Training was done only by a single operator. Future work should extend this to include training with multiple operators, but note that the pedal was trained multiple times using the same driver and consistently converged to similar policies, even under a variety of discretization schemes. The agent learned to increase pedal resistance in states between regions of certain disaster and of no risk.

5.2 Results

In general, there should be a compelling reason to give information to the human or to take action. In practice, the strongly satisficing set allows several actions to be chosen. In this experiment, the no-force action was always selected if it was in the strongly satisficing set. If not, the satisficing action that had maximal acceptability was chosen. The result was a policy that applied force only when no other option was justified.

The driving simulator consisted of a CRT, a force feedback steering wheel, a force feedback gas pedal, and a computer running a simulated driving environment. In the experiment, drivers were asked to perform a primary driving task while simultaneously performing a secondary math task. This secondary task structure simulates driver distraction, and allows us to test whether the force profile that was learned quickly from a single trainer was applicable to multiple drivers under difficult conditions.

Subjects were asked to drive two 10-minute segments, once with the active haptic forces and once with only the passive spring resistance of the pedal. During each segment, subjects answered math questions by pressing appropriate buttons on the steering wheel. The order of the control policies was randomized for each user.

The ability of the pedal algorithm to help the driver avoid crashes was well supported by the data. When a time headway value of 0.7 seconds was set as an "imminent danger" threshold, drivers spent 45% less time in the "imminent danger" zone with the haptic signal on the pedal versus without

the signal. Nine of the twelve test subjects preferred the pedal forces. Despite the users' strong preference for the pedal forces, the average NASA TLX subjective workload score only decreased from 70.65 to 70.47. Together, the expressed preferences and consistency in workload scores indicates that the system did not decrease comfort.

We conclude from this data that the learning algorithm learned a policy that helped drivers avoid risky situations without decreasing comfort. The algorithm learned this policy even though it was only trained on a single driver for fifteen minutes. This implies that the learning algorithm was able to rapidly identify forces that balanced the competing objectives of reducing risk without increasing workload.

6 Discussion

The Satisficing Q-learning algorithm produces acceptable behavior because it efficiently balances rewards and penalties. Since the true objective of a learning algorithm is to get the job done well, it is not absolutely necessary for its behavior to be produced by optimization on some performance criterion. Critics may argue that the same results could be obtained by a suitably chosen reward structure for conventional Q-learning. We agree, but we note that inventing this reward structure can be labor intensive; furthermore, the data suggests that such a structure would not be robust to variations in world structure and transition probability. This, in effect, means that the learning time is spent by the designer rather than the algorithm. We have demonstrated that dichotomous attributes can be used in a straightforward way to produce behaviors without excessive designer intervention. The key idea that allows this to be done, and a possible restriction on the algorithm, is that it produces solutions quickly for those worlds where learning what is risky or rewarding can be done quickly. We supported these arguments by demonstrating its application on a force feedback gas pedal, and by comparing algorithm efficiency against other Q-learning algorithms in simulation.

Future work should include comparing this algorithm to other approaches for doing online or fast learning. A useful metric for doing such comparisons would be the number of actions that can reasonably be explored at any given time. This work could use the notion of effective size of the exploration set as a basis for comparing information transfer. Other future work should further sensitivity analysis on parameters, such as linking the tremble decay rate to the learning decay rate.

Acknowledgements

This work was partially funded by DARPA under contract #NBCH1020013 and by Nissan Motor Company.

References

[1] J. W. Curtis and R. W. Beard. Ensuring stability of state-dependent riccati equation controller via satisficing. In *Proceedings of the IEEE Conference on Decision and Control*, pages 2645–2650, Las Vegas, NV, December 2002.

[2] M. A. Goodrich, E. R. Boer, and H. Inoue. A model of human brake initiation behavior with implications for ACC design. In *IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, pages 86–91, Tokyo, Japan, October 5-8 1999.

[3] M. A. Goodrich and M. Quigley. Learning haptic feedback for guiding driver behavior. In *Proceedings of the 2004 International Conference on Systems, Man, and Cybernetics*, Delft, The Netherlands, 2004. To appear.

[4] M. A. Goodrich, W. C. Stirling, and E. R. Boer. Satisficing revisited. *Minds and Machines*, 10:79–110, 2000.

[5] M. A. Goodrich, W. C. Stirling, and R. L. Frost. A theory of satisficing decisions and control. *IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems and Humans*, 28(6):763–779, November 1998.

[6] R. Karandikar, D. Mookherjee, D. Ray, and F. Vega-Redondo. Evolving aspirations and cooperation. *Journal of Economic Theory*, 80:292–331, 1998.

[7] S. Katayama. *Satisficing, Efficient Implementation, and Generalization for On-line Reinforcement Learning*. PhD thesis, Tokyo Institute of Technology, March 2000.

[8] T. Matsuda and S. Takatsu. Characterization of satisficing decision criterion. *Information Sciences*, 17(2):131–151, 1979.

[9] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[11] Sandip Sen, editor. *Satisficing Models*, Stanford, California, March 23-25 1998. AAAI Spring Symposium. Technical Report SS-98-05.

[12] H. A. Simon. *The Sciences of the Artificial*. MIT Press, 3rd edition, 1996.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[14] David Wingate and Kevin D. Seppi. Efficient value iteration using partitioned models. In *Proceedings of the International Conference on Machine Learning and Applications*, pages 53–59, 2003.